lbr Report No. 419
o 419
Cop 2

Math

OVERLAY SUPERVISORY PROGRAMS*

by

Vivian Wan-Man Chan

February, 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 419

OVERLAY SUPERVISORY PROGRAMS*

by

Vivian Wan-Man Chan

February, 1971

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois, 1971.

OVERLAY SUPERVISORY PROGRAMS

BY

VIVIAN WAN-MAN CHAN
B.S., University of Illinois, 1969

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1971

Urbana, Illinois

ACKNOWLEDGMENT

The author would like to express sincere gratitude to Professor H. G. Friedman, Jr. who provided guidance and helpful suggestions during the preparation of this thesis. Thanks are also due to Mr. Raymond L. Chace for his help in the project.

Miss Joyce Vaughn must be thanked for her patience in typing the manuscript.

Finally, the author would like to thank the Department of Computer Science for the facilities and staff cooperation so willingly provided.

PREFACE

The paper describes the IBM Overlay Supervisory Programs and compatible programs which use dynamic allocation of main storage. In the new programs, each segment is loaded dynamically at the time that it is required. Relocation is described in detail.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

When a program is too large to fit into available main storage at one time, it can be partitioned into segments which are brought into core for execution at the time that they are required. This technique is called overlaying.

A user of the IBM System/360 can request the loading of an overlay segment via one of the following four possibilities:

(1) A CALL macro instruction, which causes a segment to be loaded and control to be passed to a symbol defined in that segment.

(2) A branch instruction, which causes a segment to be loaded and control to be passed to a symbol defined in that segment.

(3) A segment load (SEGLD) macro instruction, which requests loading of a segment. Processing continues in the requesting segment while the requested segment is being loaded.

(4) A segment load and wait (SEGWT) macro instruction, which requests loading of a segment. Processing continues in the requesting segment after the requested segment is loaded. In each case, all segments between the requesting and the requested branches are loaded.

Although an overlay program tested out on one computer system may compile successfully on a computer system built by a different manufacturer, it is very unlikely that the program will load and execute without considerable rearrangements. Thus, it would be a tremendous saving on time and money if the different loaders can be standardized so as to eliminate the need for rearrangements.

In her paper Loader and Standardization for Overlay Programs, Miss Bernadine C. Lanzano suggested a set of features which can be used as a guideline towards standardization. One of her suggestions is a generalized tree structure which calls for more flexible utilization of memory. This scheme allows mutually exclusive overlays within and beyond the branch hierarchy. It expands on the tree structure by allowing not only lineally dependent segments but also several levels of mutually exclusive segments. Thus, when a segment is requested, the overlay supervisor will initiate loading of that segment and all the segments higher in its path, or that segment alone if it is independent.

Allocation of memory that does not impose lineal relationships on the segments would be the first step towards implementation of the overlay structure described above. With this in mind, overlay supervisory programs of the IBM 360 operating systems were studied in detail and compatible programs which use dynamic allocation of memory were written.

2. OS 360 OVERLAY SUPERVISORY PROGRAMS

Figure 1 shows the general flow of control between the different overlay supervisory programs.

The linkage editor accepts as input load modules, object modules and OVERLAY statements which define the overlay tree structure. An object module is the output of a language translator. It is in machine language, but is nonexecutable because it still contains unresolved symbols. The linkage editor generates a segment table (SEGTAB) and entry tables (ENTAB) and incorporates them into the text. Its output is a load module, which is written onto the SYSLMOD data set. A load module is in executable form because its external references have been resolved by the linkage editor.

When the operating system initiates execution of an overlay program, the contents supervisor branches to Program Fetch to load the root segment of the program. It issues a LOAD macro to bring the non-resident portion of the Overlay Supervisor into core, and then gives control to the root segment.

During execution of the root segment, a SEGLD or SEGWT macro instruction would generate an SVC 37 interruption. Control passes via Interrupt Handlers to the resident portion of the Overlay Supervisor. The resident Overlay Supervisor issues a LINK macro to the non-resident portion, which then determines the segment(s) that must be loaded. It is again Program Fetch which does the actual loading and necessary relocation. On exiting from SVC 37 interrupt, control is given to the instruction which follows the SEGLD or SEGWT instruction.

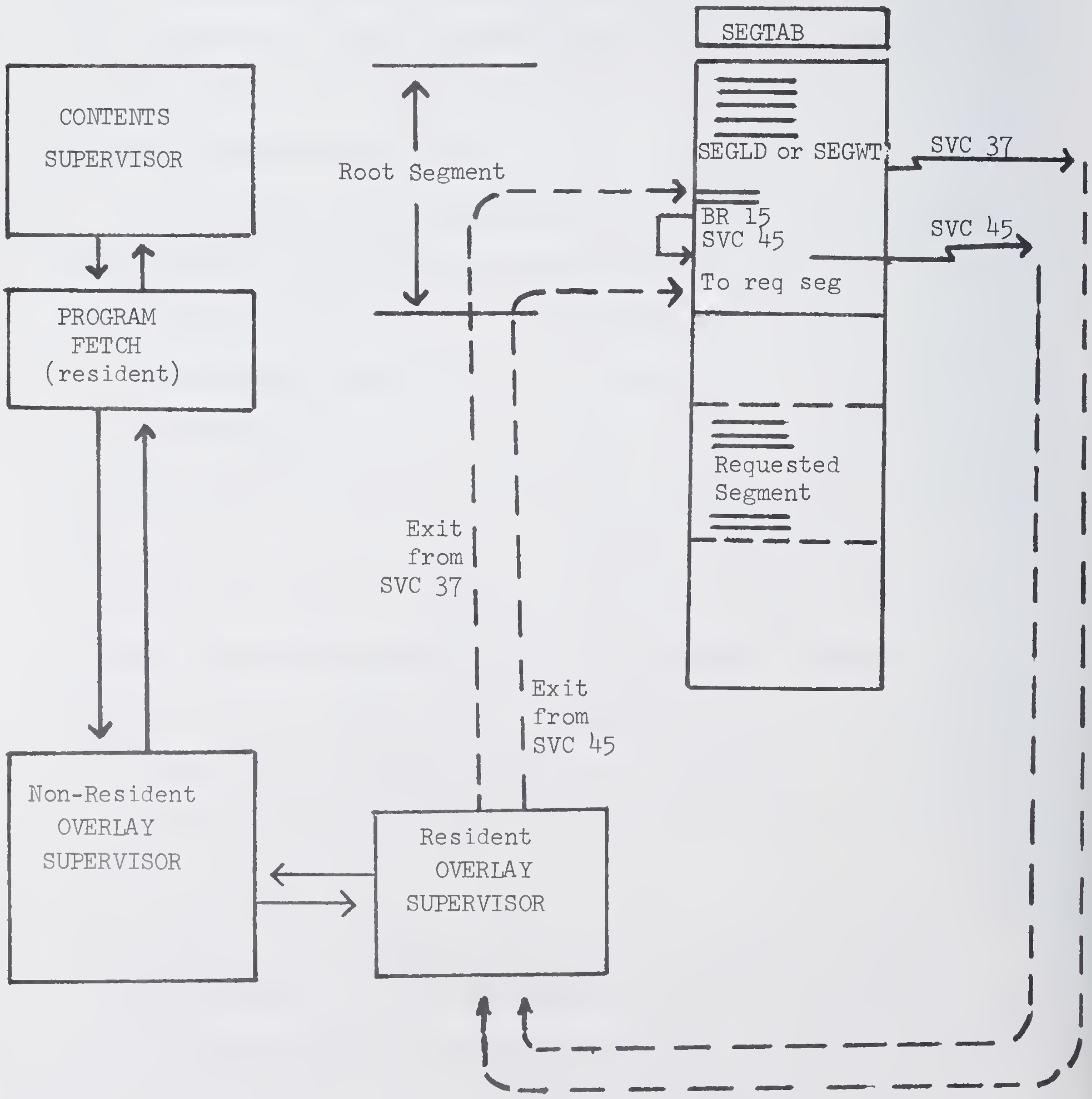


Figure 1.

Flow of Control of OS 360 Overlay Supervisory Programs

For a CALL or branch to another segment, the Linkage Editor changes the address constant so that the actual branching is to an ENTAB entry. The first four bytes of this entry are another branch instruction to the last entry of this ENTAB. From the last entry, an SVC 45 interrupt is generated. The resident Overlay Supervisor gains control via the Interrupt Handlers and proceeds as described above. Upon exit, control is returned to the last entry of ENTAB which loads the requested segment entry point into register 15 and branches to it. (See Figure 2.)

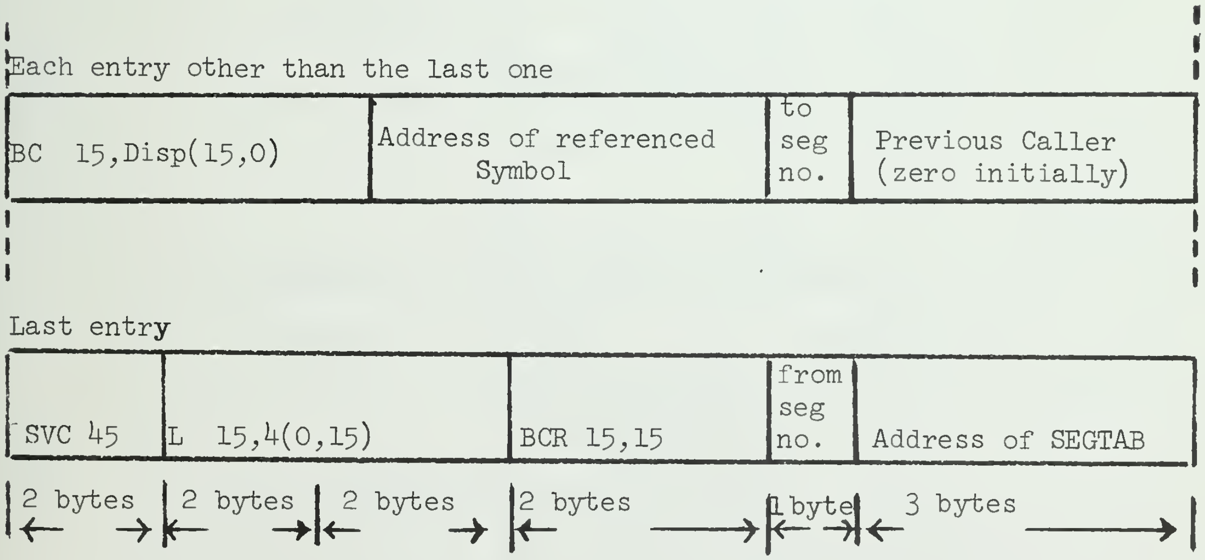
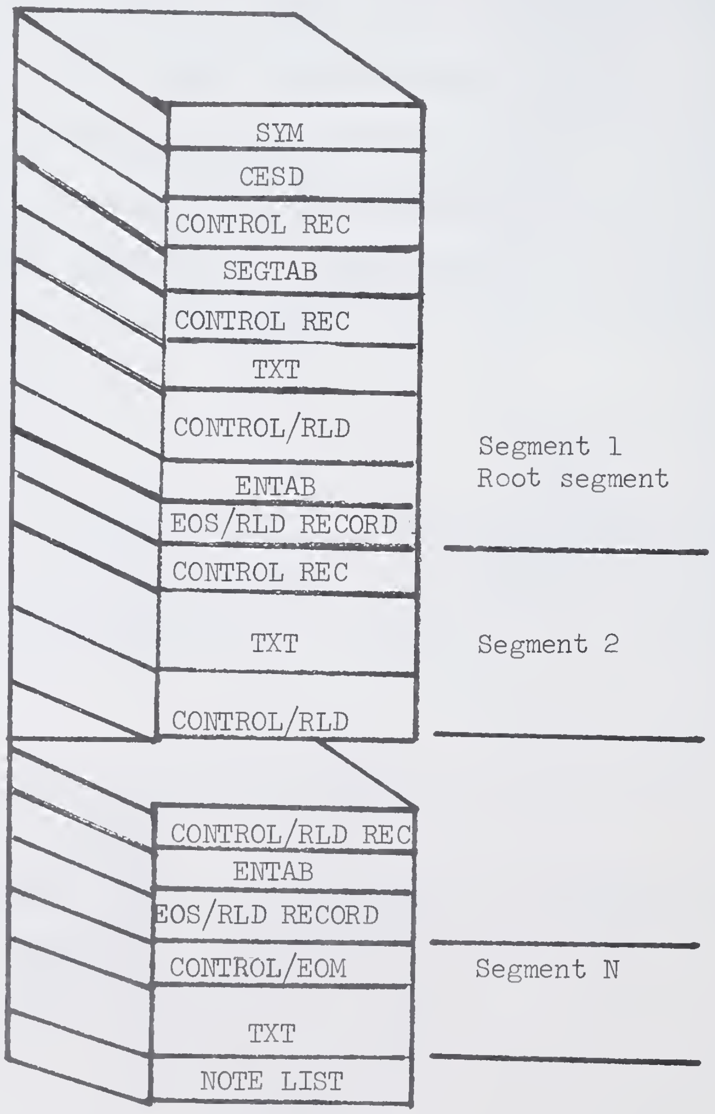


Figure 2.

The Entry Table

2.1 Load Module Structure for an Overlay Program

Figure 3 shows the logical format of a typical load module of an overlay program. The following is a description of the different types of records:*



* Refer to Reference 4 for the format of these records.

Figure 3.

Load Module Structure

Test Symbol Records (SYM)

SYM records are used by the test translator. These records are not included in the output load module unless the TEST option is specified on the EXEC statement.

Composite External Symbol Dictionary Records (CESD)

If the program is not in TEST, the CESD records always precede other records in the module. The first 8 bytes of a CESD record contain the ESD identifier (ESDID) of the first item and also the length of the record. The ESDID of a symbol indicates its relative position in the CESD. It is the line number of the symbol in the dictionary. There may be a maximum of 15 items in each record. The CESD contains entries for each named control section or common area, unnamed control section or common area, and each external symbol defined or referred to within the module. An entry is also made for each segment or entry table.

Text (TXT)

Text records contain the instructions and data of the module. The first text record in the module is the segment table (SEGTAB). SEGTAB is located in the root segment and remains in core throughout execution of the module. It is structured in such a way that only a maximum of 4 regions is allowed. There is a status indicator for each segment. The Overlay Supervisor uses SEGTAB to keep track of the relationship of various segments during execution. (See Figure 4.) An entry table (ENTAB) may be created by the Linkage Editor as the last control section of a segment. An ENTAB entry is created for a symbol to which a downward call has been made and if no other entry exists for that symbol in the caller's segment or in segments higher in the caller's path. An entry is also generated if the call is across regions. No entry is created for upward calls, which are resolved directly. If the call is exclusive and no entry exists in a common segment, the address constant is resolved directly to the symbol and usually leads to incorrect results during execution.

Control Records

Each text record is preceded by a control record or composite/RLD record which provides the channel command word necessary for reading in the text. The data address field in the CCW contains the linkage editor assigned address of the first byte of text in the text record that follows. The count field contains the length of the succeeding record.

Relocation Dictionary Records (RLD)

The relocation dictionary lists all address constants that must be relocated in the module. RLD records follow text records whose address constants require relocation. A typical RLD item consists of a relocation pointer, a position pointer, flag field

byte				
0	TEST indicator	Address of Data Control Block used to load module *		
4		Address of note list *		
8	Last segment number of region 1	Highest segment no. in storage-region 1	Last segment number of region 2	Highest segment no. in storage-region 2
12	Last segment number of region 3	Highest segment no. in storage-region 3	Last segment number of region 4	Highest segment no. in storage-region 4
16	Zero	Address of ECB to	be posted when been serviced	SEGLD request has *
20		Reserved *		
24	Previous segment no. for segment 1*			Status indctr
28	Previous segment no. for segment 2	Address of ENTAB entry (when caller chain exists)*		Status indctr
:	:			:
	Previous segment for segment N	Address of ENTAB entry (when caller chain exists)*		Status indctr

4 bytes

Description of fields:

TEST indicator Bit 1=0: Not in Test Bit 1=1: in Test; Bit 3=1: SEGLD in PROGRESS

Highest segment number in storage is initially set to 00 except for region 1 which is initially set to 01 by the linkage editor.

Status indicator indicates the status of this segment, with the two last bits of the entry table address as follows:

00 - segment is in main storage as a result of a branch to the segment.

10 - segment is in main storage, no caller chain exists.

01 - segment is not in main storage, but is scheduled to be loaded.

11 - segment is not in main storage.

The status indicator for segment 1 is initially set to 10. All the rest are initially set to 11.

* Set to zero by the linkage editor.

Figure 4.

The Segment Table

and the linkage editor assigned address of the address constant. The relocation pointer (R pointer) is the ESDID of the symbol on which the address constant depends. The position pointer (P pointer) is the ESDID of the control section which contains the address constant. The flag field specifies miscellaneous information that is necessary for relocating the address constant.

Control/RLD Records

This is a combination of the control record and the relocation dictionary record.

EOS/EOM

End of segment or end of module is indicated as a flag in the last control, RLD or control/RLD record of that segment or module.

Note List

The last section in the module is the note list. It contains the relative track address (TTR) of the first record in each segment, with the exception of segment 1. The TTR of segment 1 points to its first text record, i.e., SEGTAB. After the root segment has been loaded, Program Fetch stores in the note list the address of SEGTAB.

2.2 Memory Utilization

OS 360 expands on the tree structure by allowing several separate regions. Thus, a segment can have access to segments not in its path. This can also be used to increase loading efficiency because processing can continue in one region while the next path to be executed is being loaded in another region. This arrangement is somewhat similar to the structure suggested by Miss Lanzano, but is unfortunately limited to a maximum of 4 regions.

Although OS 360 allows multiple regions, memory utilization is very rigid. The linkage editor assigns the relocatable origin of the root segment at 0. The relative origin of each segment is 0 plus the length of all the segments in its path. The relative origin of a second region is determined by the length of the longest path in the first region. The relative origin of a third region is determined by the length of the longest path in the first region plus the longest path in the second region.

When Program Fetch receives control from the Contents Supervisor, it issues a GETMAIN macro instruction to get enough core storage for the entire load module. The root segment is loaded and it remains in core throughout execution. When a segment which is not in core is requested, it will be loaded beginning from the address which is the sum of the main storage address of the module's origin and the linkage editor assigned address of the segment. Thus, memory occupation is rather rigid and the maximum amount of storage is always assigned to a user even though he may never use the entire area at any one time. (See Figure 5.)

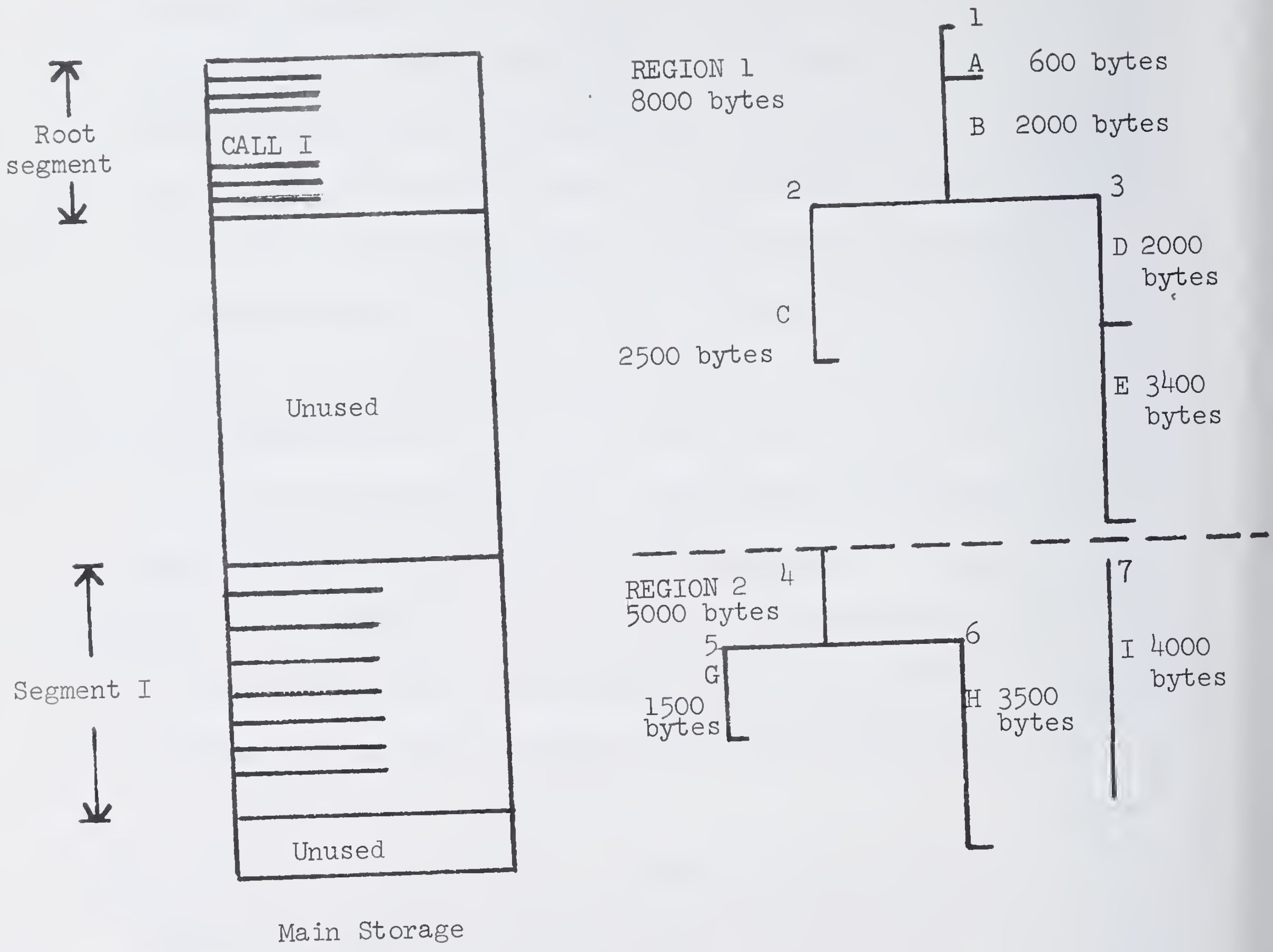


Figure 5.

Example of Memory Occupation of an Overlay Program

2.3 Program Fetch*

Program Fetch resides in the system nucleus and is invoked by the Contents Supervisor or the Overlay Supervisor. It is invoked by the Contents Supervisor after a LINK, LOAD or XCTL macro instruction has been issued for a module which is not in main storage. It is invoked by the Overlay Supervisor after a SEGLD, SEGWT or CALL macro instruction or a branch instruction if the needed segment is not in main storage.

The major functions of the Program Fetch routine for the loading of an overlay program are:

INITIALIZATION

Program Fetch initializes a fetch work area whose address is supplied by the caller. It places in this area information that is later used for loading the requested segment. It sets up in this area an I/O control block that is needed by the I/O Supervisor, event control blocks, 3 channel programs to overlap the reading of records with relocation, 3 RLD buffers and a buffer table which contains a 12-byte entry for each RLD buffer. If the request is from the Contents Supervisor, Program Fetch also receives the pointer to the partitioned organization directory record that is built by the BLDL routine.

* Only the aspects of FETCH dealing with overlay programs are described.

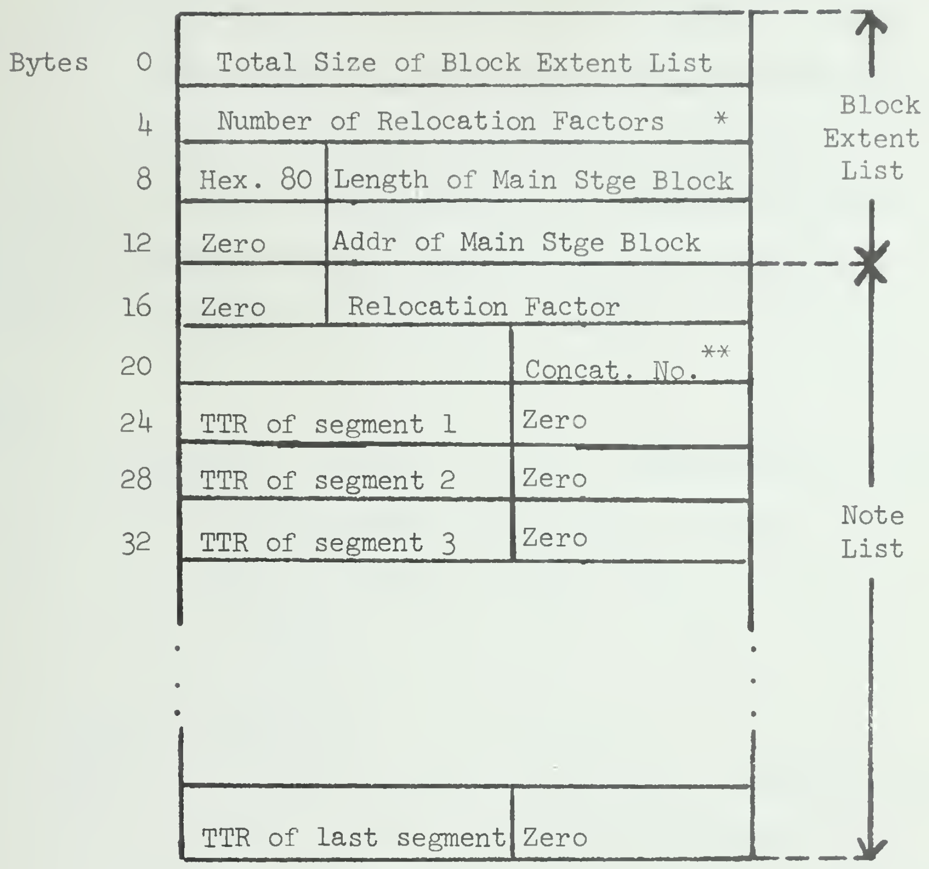
From the information given, Program Fetch can get space for the extent list and note list. It also issues a GETMAIN macro instruction to obtain space for the entire load module. Program Fetch reads in the note list and then proceeds to set up the extent list. (See Figure 6.)

LOADING

With the relative track address supplied by the note list, Program Fetch calls on the system 'convert' routine to change it into an absolute disk seek address. The I/O Supervisor is invoked to start I/O operations at the indicated address with an EXCP macro instruction. Control, RLD or Control/RLD records are read into the RLD buffers. Text records are read into main storage, beginning at the first assigned main storage address contained in the extent list. After the channel program has been started, the I/O Supervisor returns control to the Program Fetch routine to await posting of an event control block by the I/O Supervisor or an appendage routine. Such posting indicates that one or two records have been read and that further processing can occur in the Program Fetch routine.

RELOCATION

The Program Fetch routine is restarted after the PCI appendage routine or the I/O Supervisor has posted an ECB. Then the Relocation subroutine of Program



* Set to 01 by Program Fetch

** Concatenation number is a value that specifies this data set's sequential position in a group of concatenated data sets.

Figure 6.

Block Extent List and Note List

Fetch examines the buffer table to determine if RLD record is in an RLD buffer. When such a buffer is found, the Relocation subroutine relocates each address constant found in that record. After the buffers are processed, the Program Fetch routine restarts a channel program or awaits the posting of the loading of another record.

The relocation constant is the difference between the linkage editor assigned origin of the module (usually zero) and the absolute address of the origin of the module. An overlay program is block loaded. The position constant is the same as the relocation constant. A flag in each RLD item indicates whether the relocation contant is to be added to or subtracted from the value field of each relocatable address constant. As an example, assume that a module is loaded beginning at address 4000. A position constant of 4000 is added to the address field of the RLD item to get the absolute address of the address constant that is to be relocated. The value field of the address constant is loaded and if the flag bit in the RLD record is positive, a relocation constant of 4000 is added to it and then it is restored in its original position.

Since the relative position of each segment in the module has been determined by the linkage editor, the 'address of referred to symbol' field in each

ENTAB entry contains the linkage editor assigned address of the symbol at an offset from the linkage editor assigned origin of that module. After relocation, this field contains the actual address where the symbol will be located after the segment in which it resides has been loaded.

TERMINATION

If Program Fetch is called by the Contents Supervisor, it places the main storage address of the DCB and of the note list into SEGTAB. Then, control is returned to the caller.

2.4 The Overlay Supervisor

RESIDENT MODULE: The resident Overlay Supervisor resides in the system nucleus. It is given control by the SVC Interrupt Handlers for SVC 37 and SVC 45. The status of the requested segment is examined. If it is already in main storage and the ENTAB entry is prepared for direct branching, the Overlay Supervisor returns control to the next instruction in ENTAB. If the requested segment is not in main storage, the resident Overlay Supervisor links to the non-resident Overlay Supervisor under the name IEWSZOVR.

Upon return to the module, a return code is examined and if an error has been detected, an ABEND is issued.

NON-RESIDENT MODULE (IEWSWOVR) The non-resident Overlay Supervisor is loaded by the Contents Supervisor. The name of this module is IEWSWOVR, its member name is IEWSZOVR, and it is referenced under its member name. It is linked to by the resident Overlay Supervisor when an overlay segment issues a SEGLD or SEGWT request for another segment, or when a segment issues a CALL or branch instruction to an external address of another segment not in main storage.

The operations performed by the non-resident Overlay Supervisor depend on the type of instruction issued by the caller and the status of the required segment.

<u>REQUESTED SEGMENT IS IN MAIN STORAGE</u>: If the request is a SEGLD or SEGWT, control is returned to the caller. If the request is a CALL or branch instruction, the ENTAB entry is altered to prepare future unassisted branch to the segment, bypassing the SVC 45 of the caller's ENTAB. The value 2 is added to the displacement field of the ENTAB entry through which the branch of the SVC 45 instruction was routed. (See Figure 2.) When the caller executes another branch to this ENTAB entry, control will be given to the second field of the last ENTAB entry. Execution of the instruction in this field will cause general register 15 to be loaded with the value assigned to the address constant. A branch to that location in the requested segment will then be executed.

<u>REQUESTED SEGMENT IS NOT IN MAIN STORAGE</u>:

REQUEST IS SEGLD: The SEGLD flag in SEGTAB is turned on to indicate a segment load is in progress. The requested segment and segments in its path are determined and marked for loading. The Overlay Supervisor gets space for a fetch work area. An extra 100 bytes is obtained for saving the registers through ATTACH. An IDENTIFY macro is issued to add the entry point IEWSEGLD to the module in execution. Then it attaches to that entry point as a subtask. Control is returned to the caller. The attached routine branches to the Program Fetch routine to load the marked segments. After the segments are loaded, the attached routine posts the event control block in SEGTAB.

REQUEST IS SEGWT: Requested segment and segments in its path are marked for loading. The Overlay Supervisor gets space for the fetch work area and then branches to the Program Fetch routine. Control is returned to the caller only after all marked segments are loaded.

If the requested segment is being loaded for a previous SEGLD request, the Overlay Supervisor issues a WAIT macro to await completion of the loading.

REQUEST IS CALL OR BRANCH: Requested segment and segments in its path are marked for loading.

A 1200-byte work area is obtained and the Overlay Supervisor branches to Program Fetch to load the marked segments. If the requested segment has been requested previously via a SEGLD and loading is not complete, the Overlay Supervisor issues a WAIT macro to await the posting of the ECB. In either case, the caller's entry table is altered to prepare for any future unassisted branch to the same external address. Then control is given to the requested segment at the specified address.

DETERMINING THE SEGMENTS THAT MUST BE LOADED: There is a 4-byte entry for each segment in SEGTAB. Each entry contains the number of the preceding segment in the path and a field of status indicators. (See Figure 4.) The number of the highest segment in main storage for the region of the requested segment is compared to the requested segment number. If the new segment number is greater than the old one, then it is marked to be loaded. The preceding segment in its path is now called the new segment and the step is repeated.

If the old segment number is greater than the new one, then the old segment is being overwritten. Tables for segments being overwritten are reset. The preceding segment in the path of the old segment is set to be the old segment and the above step is repeated.
If the old segment number is equal to the new segment number, the Overlay Supervisor prepares for entering Program Fetch to load the marked segments.

TERMINATION: Control is returned to the resident Overlay Supervisor. Upon exit, general register 15 is set to a return code which is interpreted by the resident Overlay Supervisor. On normal returns, the code is zero.

3. OVERLAY SUPERVISORY PROGRAMS USING DYNAMIC ALLOCATION

Since the Program Fetch routine obtains space for the entire load module when called by the Contents Supervisor, it is necessary to rewrite both the Program Fetch routine and the non-resident Overlay Supervisor in order to allocate main storage for each segment dynamically. Two separate modules, OVFETCH and IEWSZOVR, functionally replace the Program Fetch routine and the non-resident Overlay Supervisor, respectively. These modules are located in a private program library called &&MYLIB.

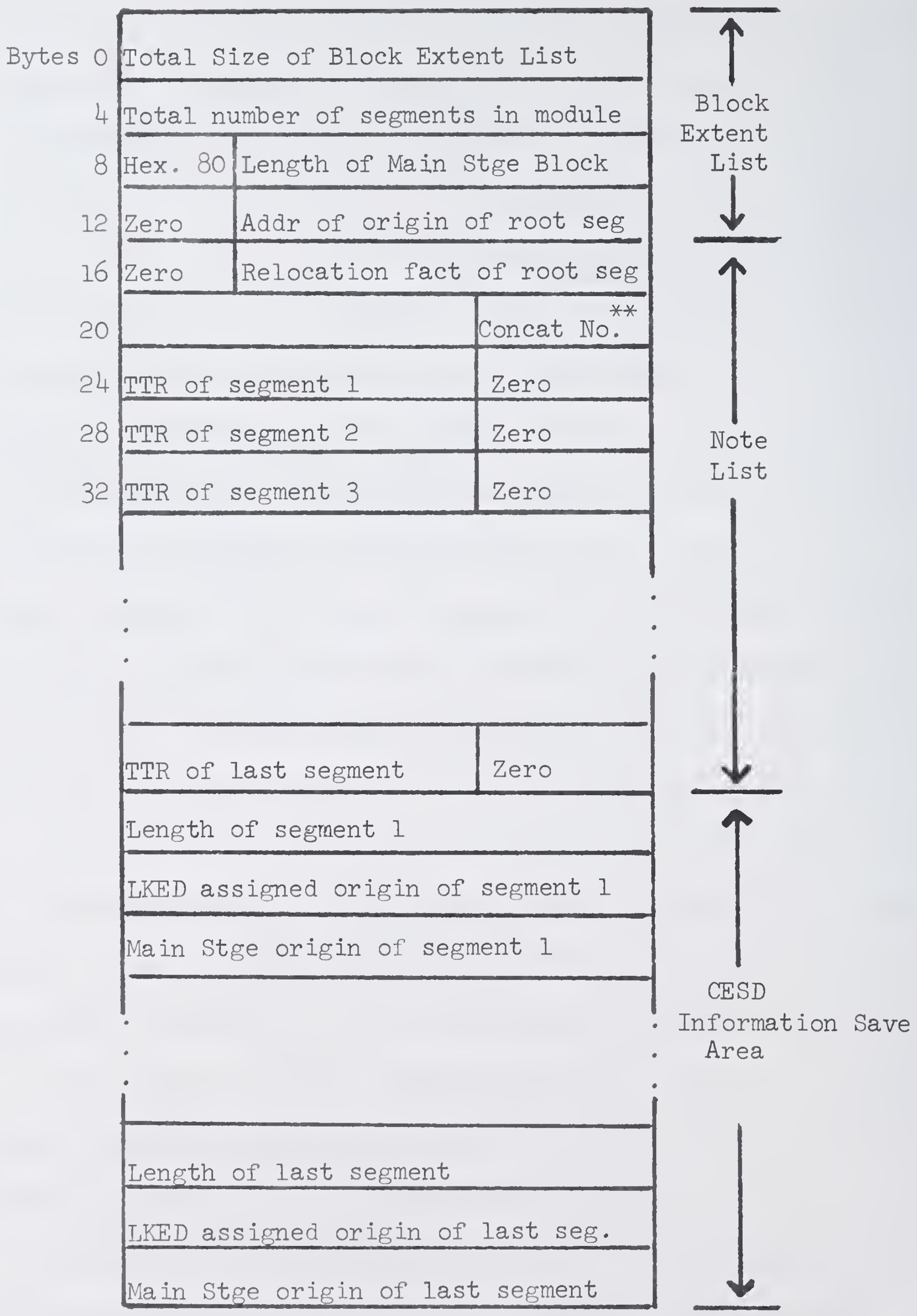
The name IEWSZOVR is chosen for the Overlay Supervisor which uses dynamic allocation because this is the name to which the resident Overlay Supervisor links. At the GO step, a STEPLIB card is used to effectively concatenate &&MYLIB with SYS1.LINKLIB, the system library. When the loading of a module is requested, the system first searches for it in the specified private library. If it is not found, the system searches for it in the system library. Thus, it is this Overlay Supervisor which gains control from the resident Overlay Supervisor instead of IEWSWOVR, the IBM non-resident Overlay Supervisor.

To perform the functions of the Contents Supervisor for an overlay program, a simple loader, OVLDER, was written. This is essentially the driver program which initiates execution and loads the other two modules. The following is a description of the three modules which were written and tested. These modules are compatible with their IBM counterparts.

3.1 The Overlay Loader (OVLDER)

This is the module which is first initiated for execution. It assumes that the user's load module is on SYSLMOD and that it has by default, the name &&GOFILE(MAIN). It contains the data control block (DCB) for reading in the user's program. A BLDL macro instruction is issued to obtain information concerning the user's module. Then it gets space for the extent list, the note list and the CESD information save area described below.

Since each segment is to be loaded dynamically at the time that it is needed, it is necessary to gather information from the CESD records so as to determine the length of each segment. The linkage editor assigned address of the origin of each segment is also needed for relocation purposes. A CESD information save area is tagged onto the end of the note list to store this information. (See Figure 7.) Thus, when the non-resident Overlay Supervisor receives control, it can easily determine the pointer to this area from the pointer to the note list which is passed along from the resident Overlay Supervisor. The extent list and the note list contain the same information as before, except for the eighth byte of the note list which contains the total number of segments in the module. In


Bytes 0
Total Size of Block Extent List
4
Total number of segments in module
8
Hex. 80
Length of Main Stge Block
12
Zero
Addr of origin of root seg
16
Zero
Relocation fact of root seg
20
Concat No.**
24
TTR of segment 1
Zero
28
TTR of segment 2
Zero
32
TTR of segment 3
Zero
TTR of last segment
Zero
Length of segment 1
LKED assigned origin of segment 1
Main Stge origin of segment 1
Length of last segment
LKED assigned origin of last seg.
Main Stge origin of last segment
Block Extent List
Note List
CESD Information Save Area

**Concatenation number is a value that specifies this data set's sequential position in a group of concatenated data sets.

Figure 7.

Extent List, Note List and CESD Information Save Area the CESD information save area, a 12-byte entry is created for each segment. The length fields are initially set to zero's. When a CESD record is read, each item is checked so that if it is a control section, a common save area or a private code marked delete (SEGTAB or ENTAB), its length in bytes (or next higher multiple of eight if not a multiple of eight) is added to the length field of the segment it is in. The linkage editor assigned origin of each segment is also saved. Then the Overlay Loader sets up the extent list and reads in the note list.

OVFETCH and IEWSZOVR are loaded by means of LOAD macro instructions, and the entry point of OVFETCH is stored into IEWSZOVR. Then the Overlay Loader sets up the appropriate registers and branches to OVFETCH to load the root segment.

Upon exit from OVFETCH, register 15 contains the completion code. On normal completions, register 4 contains the entry point to the root segment. Control is given to the root segment. If an error is detected while loading the root segment, an error message is printed and execution is terminated.

3.2 <u>The Non-Resident Overlay Supervisor</u> (IEWSZOVR)

This module is very similar to the IBM non-resident Overlay Supervisor except for a few critical differences:

(1) Since main storage for each segment is to be allocated dynamically, it is necessary to free main storage for the segments being overwritten. This section of codes is added

to the Overlay Supervisor because it is the latter which rewrites the tables for the segments being overwritten. From the CESD information area, the Overlay Supervisor loads the length and main storage origin of the segment to be freed and issues a FREEMAIN macro instruction.

(2) For a CALL or branch instruction, an ENTAB entry pointer is given. The value of the address field in this entry is merely the sum of the linkage editor assigned address of this external symbol and the main storage origin of the root segment. (See section on RELOCATION in 3.3.) Thus, it is necessary to relocate this address constant before branching back to the caller.

This relocation is performed at the termination section of the Overlay Supervisor because the segment containing the requested external symbol must have been loaded by then and its main storage origin is available. The relocation can be given by this formula:

new address constant = address constant in ENTAB (i.e., linkage editor assigned address + main storage origin of root segment)
- main storage origin of root segment
+ main storage origin of the segment containing the external symbol
- linkage editor assigned origin

of the segment containing the external symbol.

This new address constant is the actual address of the external symbol in main storage. It is stored in the ENTAB entry and processing continues.

For example, assume that the root segment is loaded at location 1000, the linkage editor assigned origin of segment 3 is 300 and that a CALL to an external symbol at an offset of 40 bytes from the origin of segment 3 is executed. Further assume that segment 3 is loaded beginning from location 4000. Before relocation in OVFETCH, the address field of the ENTAB entry contains 340. After relocation in OVFETCH, the address field of the ENTAB entry contains 1000+340=1340. Applying the above formula, the final relocated constant is 1340-1000+4000-300=4040.

3.3 <u>The Overlay Fetch Program</u> (OVFETCH)

OVFETCH gets space for a requested segment and loads it into main storage. It is not as efficient as the IBM Program Fetch routine because READ macros are used instead of EXCP macros. However, a version using EXCP could be written.

OVFETCH gets the length of the requested segment from the CESD information save area, issues a GETMAIN macro instruction for the needed main storage and stores the segment entry point into the CESD information save area. When OVFETCH receives control from the Overlay Supervisor, it is in supervisory state with protection key zero, so a tracing of task control block(s) is necessary to get the user's key. Then, a set storage key instruction is used to assign the user's protection key to the newly obtained block of storage. Because OVFETCH is in supervisory state, a subpool ID of 250 is specified so as to obtain core storage from subpool 0. If the request is from the Overlay Loader, OVFETCH is not in supervisory state, and core storage is obtained from subpool 126.

Text records are read into the assigned main storage area. RLD, control and control/RLD records are read into the RLD buffer that is in the fetch work area. READ macros are used for reading in the records.

RELOCATION: The main problem is, of course, with relocation. If the segment just loaded is the root segment, relocation proceeds as in the IBM Program Fetch routine, i.e., relocation constant = position constant = main storage address of the origin of the root segment.

If the loaded segment is not the root segment, relocation proceeds as follows:

The position constant is the main storage origin of the loaded segment minus the linkage editor assigned origin of the SEGMENT. The address field in the RLD item is added to the position constant to obtain the actual address of the address constant that is to be relocated. The address constant thus obtained is relocated following these steps:

Let ADCON be the value of the address constant that is to be relocated.

Let n be the number of the segment in which the external symbol pointed to by ADCON resides.

(1) Set n = 1.

(2) If ADCON is less than the linkage editor assigned origin of segment n then go to (7).

(3) If ADCON is greater than or equal to the sum of the linkage editor assigned origin of the segment n and its length then go to (7).

(4) The status indicator of segment n is checked. If segment n is marked not in main storage and not to be loaded then go to (7).

(5) The relocation constant is the main storage origin of segment n minus the linkage editor assigned origin of segment n.

(6) The relocation address constant is ADCON + the relocation constant. Exit.

(7) n = n + 1.

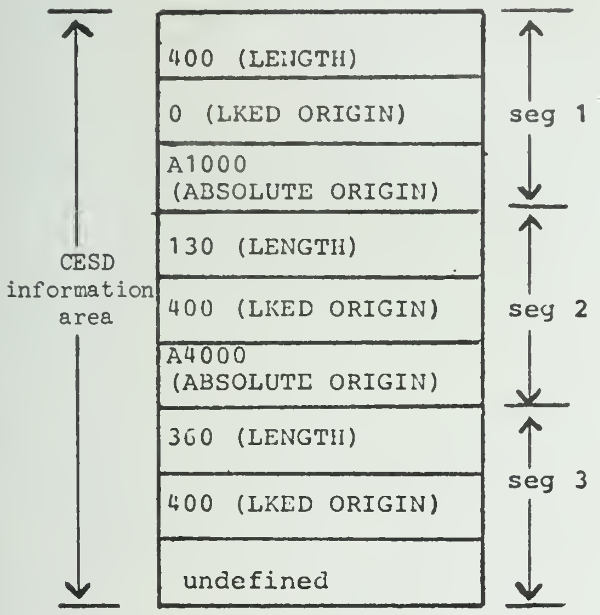
(8) If n is less than or equal to the total number of segments in this module, go to (2).

(9) The relocation constant is set to the main storage origin of the root segment. Go to (6).

The segment in which ADCON resides must be determined because the position of the former is independent of the position of the segment that has just been loaded. Thus, the main storage origin of this segment must be determined in order to relocate ADCON. The linkage editor assigned address, ADCON, is compared to the entries in the CESD information save area in turn until it falls in the range of a segment which is in main storage or is marked to be loaded. If such a segment is found, the relocation constant is set to the value of the linkage editor assigned origin of that segment subtracted from its main storage origin. If the found segment is marked to be loaded, it is the segment that has just been loaded. An example of this type is the address field of a branch instruction to an ENTAB entry which is within the same segment.

If the address constant is within the range of a segment which is neither in storage nor marked to be loaded, that segment is mutually exclusive to the one in which the address constant resides, so the scan continues.

If all the entries have been checked and the condition is still not satisfied, the address constant to be relocated is the value of an address field of an ENTAB in the loaded segment. The relocation constant is set to the main storage origin of the root segment and the address constant is relocated accordingly. This is not the real address of the external symbol, but since the segment in which external symbol resides has not yet been loaded, there is no way to determine its absolute address at this point. Value fields of ENTAB entries are relocated by the non-resident Overlay Supervisor after the segment has been loaded. (See Figure 8 for an example on RELOCATION.)



RLD item:

P	R	Flag	Address
2	2	ID	440

Position constant = A4000 - 400 = A3600

Address constant is in location A3600 + 440 = A4040

Address constant loaded from location A4040 and is 500.

Relocation constant = A4000 - 400 since 400<500<230+400

Relocated constant = 500 + A4000 - 400 = A4100

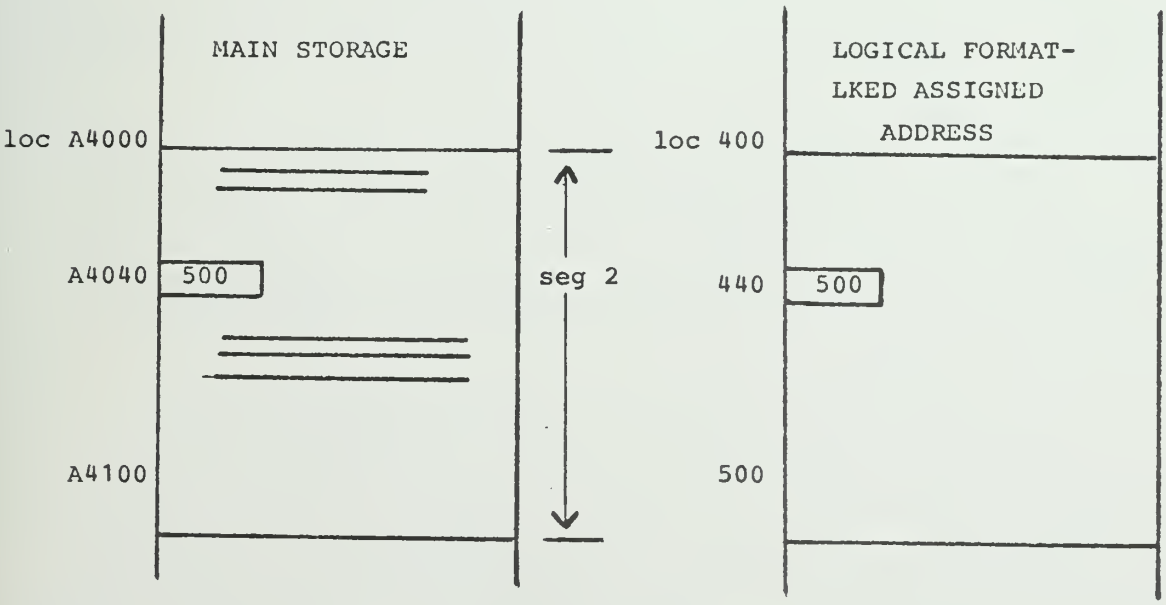


Figure 8.

An Example on Relocation

4. CONCLUSION

The modules described in chapter 3 serve as a prototype for the standardization of Overlay Supervisory Programs. Relocation has been described in detail and this method of relocating dynamically loaded segments may prove to be useful for computer systems other than the IBM 360 computers.

These modules are not as efficient as their IBM counterparts because a GETMAIN macro instruction is issued for the loading of each requested segment and a FREEMAIN macro instruction is issued for each segment being overwritten, and these instructions are relatively slow. In addition, READ macros are considerably slower than EXCP macros in reading the records, but this deficiency could be eliminated.

However, with dynamic allocation, only the minimum amount of storage is used at all times. A tree structure is no longer rigidly imposed, and further modifications are facilitated.

The next step towards the implementation of the structure described in chapter 1 would be the elimination of a limit on the number of regions. This has to be done at the linkage editor level because it is then that SEGTAB is constructed. (See Figure 4.) It is proposed that the area which records information on a region's last segment number and highest segment number in main storage be made dynamic so as to accommodate any number of regions. With that and an additional byte of information, say at the fourth byte, to record the total number of regions in the module, the Overlay Supervisor can easily be altered to check for any number of regions.

BIBLIOGRAPHY

IBM Systems Reference Library. IBM System/360 Operating System Linkage Editor (F), Program Logic Manual. File No. S360-31, Form Y28-6667-0, First Edition, January 1968.

IBM Systems Reference Library. IBM System/360 Operating System Linkage Editor and Loader, File No. S360-31, Form C28-6538-7, Eighth Edition, July 1969.

IBM Systems Reference Library. IBM System/360 Operating System Loader, Program Logic Manual. File No. S360-31, Form Y28-6714-0, First Edition, May 1969.

IBM Systems Reference Library. IBM System/360 Operating System MVT Supervisor, Program Logic Manual. File No. S360-36, Form Y28-6659-3, Fourth Edition, November 1968.

B. C. Lanzano. "Loader Standardization for Overlay Programs," Communications of ACM, Volume 12 Page 541, 1969.